A Preliminary Analysis of Recursively Generated Networks

Eric Mjolsness

David H. Sharp

Research Report YALEU/DCS/RR-476
May 1986

A Preliminary Analysis of Recursively Generated Networks

Eric Mjolsness
Department of Computer Science
Yale University
New Haven, CT 06520

David H. Sharp
Theoretical Division
Los Alamos National Laboratory
Los Alamos, N.M. 87545

Abstract

This paper outlines an approach to the automated generation of structured networks of threshold elements or "neurons" for computational purposes. The structured networks are obtained by recursive operations on primitive elements, constrained to produce networks whose recursive description is relatively simple. A preliminary analysis of the performance of structured networks on a specific continuous coding problem is presented, and compared to the performance of unstructured networks on the same problem. This comparison shows that recursively structured networks can exhibit efficient learning and high performance, while permitting generalization to larger instances of the same problem.

I. Introduction

Along with the recent interest in networks of threshold elements ("neural" networks) as a way of expressing and implementing parallel computations, there is an understandable tendency to approach the design of such networks with some of the same methods that have proven useful in analysing individual computing networks: analogies with simple dynamical systems, collective phenomena, and algebraic objective functions (e.g. Hinton and Sejnowski[1], Lapedes and Farber[2]). There are also well-established methods for designing circuits, algorithms, software and computer systems which are based on principles of hierarchical design. We aim to combine these apparently disparate approaches, mapping hierarchical design to scaling ideas (similar to the renormalization group formalism described in Wilson and Kogut[3]) for dynamical systems, in the context of an especially simple computation.

The use of hierarchical design leads to structured networks, that is ones describable with only a few parameters, generated by recursive operations on primitive elements. Given a structured and an unstructured network which perform the same computation, we prefer the structured one since it is easier to discover (being an element of a much smaller search space) and it is more likely to suggest a generalization to larger network and problem sizes. If such generalization can be automated, it would provide an interesting example of dynamical learning.

In this paper we generate structured networks which solve a variation of an encoding problem considered by Hinton and Sejnowski[1]. This problem is formulated in section II, and section III contains a description of the search strategies used to obtain several types of structured networks. In section IV, the performance of structured and unstructured networks is compared. The comparison involves both testing the networks' performance on the encoding problem and, to evaluate the prospects for dynamical learning in this task, examining the networks' generalizability to larger problem sizes. The results support the idea that it is possible to find a dynamical method for adjusting the free parameters of a structured threshold network which is capable of solving (or learning to solve) the test problem and which is sufficiently simple that it could plausibly be tried out on other problems as well.

II. A Continuous Coding Problem

We consider the following problem. A unary input (a picture-like representation of one number) is to be converted into some code which has extensive continuity properties: a small change in the input number corresponds to a small Hamming distance between the two output codes (as in a Gray code), a small Hamming distance between two code words corresponds to a small difference between the corresponding numbers (graceful degradation under code word corruption – a kind of fault tolerance), and in general the unsigned difference between two numbers is to be a monotonic function of the Hamming distance between the two codes. If $\{s_i = 0 \text{ or } 1\}$ (with only one $s_i = 1$ at a time) is a set of N threshold elements representing the unary input, and $\{s_\alpha = \pm 1\}$ is a set of $A = O(\log N)$ threshold elements representing the code word output, then

$$s_\alpha = \mathrm{sgn}(\sum_i F_{\alpha i} s_i) \qquad = \sum_i F_{\alpha i} s_i, \text{ if } F_{\alpha i} = \pm 1$$

represents both the threshold network's operation and the code. This is because, if $s_i = 1$ and $s_j = 0$ for $j \neq i$, the output code word is $s_\alpha = F_{\alpha i}$. (We assume that $F_{\alpha i} = \pm 1$.) Thus the columns F_{*i} of F are the possible code words corresponding to the possible inputs i, $1 \leq i \leq N$. Likewise, each output element s_α may be thought of as a "feature detector". The rows $F_{\alpha *}$ of F are the features which may be detected; in other words, they are the sets of input values i which would excite the given output element. The desired continuity and monotonicity properties are enforced by minimizing the function

$$E = \frac{1}{N^2} \sum_{ij} \Big[\Big| \frac{i-j}{N} \Big|^p - \frac{1}{A} d_{\mathrm{Hamming}}(F_{*i}, F_{*j}) \Big]^2 \tag{1}$$

for $p > 0$. Since the Hamming distance between two code words is

$$d_{\text{Hamming}}(F_{*i}, F_{*j}) = \frac{1}{2}\sum_{\alpha=1}^{A}(1 - F_{\alpha i}F_{\alpha j})$$

we have

$$\begin{aligned} E(F) &= \frac{1}{A}\sum_{\alpha=1}^{A}\frac{1}{N^2}\sum_{i,j=1}^{N} F_{\alpha i}F_{\alpha j}\left(\left|\frac{i-j}{N}\right|^p - \frac{1}{2}\right) \\ &\quad + \frac{1}{4A^2}\sum_{\alpha,\beta=1}^{A}\left(\frac{1}{N}\sum_{i=1}^{N}F_{\alpha i}F_{\beta i}\right)^2 + \text{constant} \qquad (2) \\ &= \sum_{\alpha} E_1^{(p)}(F_{\alpha *}) + \sum_{\alpha\beta} E_2(F_{\alpha *}, F_{\beta *}) + \text{constant} \end{aligned}$$

and as indicated E may be divided into a single-feature term and a feature-interaction term. The latter term is minimized for orthogonal features.

Aside from being a useful test case, this encoding problem could be of some interest for non-automatic network design: one is trying to represent a (possibly) meaningful quantity by a correlation (a Hamming distance), and correlations are easily computed with threshold element networks.

To minimize $E(F)$, the most direct approach would be to use a hill-climbing or simulated annealing algorithm. The disadvantages of these methods in general are that they produce irregular networks (hard to understand and to generalize to much bigger networks) and that they are often unacceptably slow. As an alternative, we will examine very compact recursive descriptions of the feature matrix $F_{\alpha i}$, and we will minimize E as a function of these recursive descriptions.

III. Recursively Generated Networks

We will introduce two different kinds of concise recursive descriptions of feature matrices. The second is more elaborate and optimized than the first, but large improvements remain to be made here. In fact, we immediately restrict ourselves to recursive descriptions of the rows of the feature matrix, or "features", $F_{\alpha *}$.

For example, a feature may be recursively built up by duplication (d_+) and duplication with negation (d_-):

$$F_{\alpha *}^{(1)} = (+1)$$

$$F_{\alpha *}^{(2)} = d_+(+1) = (+1, +1)$$

$$F_{\alpha *}^{(3)} = d_- d_+(+1) = (+1, +1, -1, -1)$$

$$d_+ d_- d_+(+) = \; + + - - + + - - \; \text{(shorter notation)}$$

so that a feature with N signs is represented by only $\log N$ signs ($d_\pm$), with great economy. The set of feature vectors so generated corresponds to the Walsh functions, and we would like to find those $A = O(\log N)$ out of N Walsh functions which minimize $E(F)$. With luck, there will be a pattern which generalizes to larger N and A. For example the binary encoding is represented by a set of features

$$F_{\alpha*} = d_+^a d_- d_+^b, \qquad a+b+1 = \log_2 N$$

and a binary-reflected Gray code has

$$F_{\alpha*} = d_+^a d_- d_- d_+^b, \qquad a+b+2 = \log_2 N.$$

The Walsh functions represent features with such economy that it is possible to exhaustively search all the features expressible with d_+ and d_-; there are only N of them. Since the recursive descriptions have been applied only to the features $F_{\alpha*}$, and not to the entire matrix F, an ad hoc search method for feature matrices was used in place of exhaustive search which would take N^A steps – the usual combinatorial explosion which we are able to avoid (only) for single features.

Instead, a beam search of beam width w was used to build up a set of feature matrices (the best w matrices found so far) one feature at a time. The procedure involved trying out each of the N possible concisely-described features as the addition to each of w matrices and keeping, out of the resulting Nw candidate matrices, the w matrices of lowest score $E(F)$. Finally, the beam width w was varied somewhat and the experiment was repeated, to insure that the results were not substantially affected by the beam width.

This procedure forces a serious asymmetry between the α and i indices of F (excused perhaps by the asymmetric index ranges: $A = O(\log N)$) and is thus just a first step towards a dynamics of recursive network descriptions; the α index is not described recursively.

The second kind of recursive description allows, in addition to the $d_\pm$ operations, the concatenation of two features whose relationship is more distant: they share common subfeatures. We consider concatenation with and without negation, producing part-sharing trees as in Figure 1. (This figure shows a feature which occured in the best network produced by the search procedure discussed below, for $p = .5$, $N = 32$, and $A = 10$.) In order to prevent a combinatorial explosion in the number of such trees considered, the subfeatures to be concatenated (the "parts" of features) are forced to be shared by the requirement that the tree structure have small width at each level. In detail, the part-sharing trees are put into equivalence classes based on the sequence of numbers of nodes per level $(a_1, b_1, \ldots a_l, b_l)$, and only those equivalence classes of cardinality less than some threshold are kept:

$$\text{cardinality} = \prod_{k=1}^{l} 2^{b_k - 1} S_{a_k}^{(b_k)} < \text{maximum}$$

$$S_a^{(b)} = \text{Stirling number of the second kind}$$

$$= \text{number of partitions of } a \text{ elements into } b \text{ groups}$$

This rule may be efficiently implemented as tree-pruning applied during the recursive construction of the allowed features. Features are combined using a beam search as before, and it must be checked that neither the beam width nor the cardinality threshold affect the results.

At this point the outline of the part-sharing trees experiment should be clear, but there is an important modification which vastly speeds up the search process with little or no degradation in performance as measured by $E(F)$. The modification involves constructing the trees from the top down, as before, but testing the partially built trees and removing from consideration those of relatively poor single-feature score. (A more complicated criterion, based on the frequency of occurrence of a partially built feature inside of low-scoring feature matrices, did not help much.) In what follows, at generations $k = 1, 2, 3, 4$, and 5 the fraction of the features kept was 1, 1, .5, .3, and .3, respectively; these fractions seem to be about the minimum necessary to retain the optimal feature matrices.

This modification is a plausible one: the bottom portions of a part-sharing tree consist of "shared parts", usually large blocks being built up, and the top regions of such a tree are "templates" for the coarse organization of a feature. We therefore find that it is possible to build good features using relatively few templates whose merit has been shown in previous, smaller encoding networks. In this sense the network "learns" from previous experience with easier problems.

IV. Results

The minimal scores $E(F)$ obtained by hill-climbing, by the semi-recursive network search procedure for Walsh features, and by the more sophisticated semi-recursive search procedure for part-sharing trees are compared in Figure 2. All three methods are comparable for $p <\approx .3$; outside this range hill-climbing is superior to the Walsh functions as well as being much faster as described below. Furthermore, the part-sharing trees score remains very close to that of hill-climbing over most of the range of p (for $0 < p <\approx .7$). The most surprising result, however, is the extreme orderliness of the Walsh function semi-recursive networks. Figure 3 shows this for $p = .01$ (or any p sufficiently near to 0): features are ranked by the number of same-sign blocks (or the number of sign-changes as i varies), and the top A features are chosen to produce the optimal semi-recursive network. This ranking may also be obtained by examining the single-feature scores, a point of some interest for improving the dynamical system which is to produce F. For $p \neq 0$ the pattern of the optimal semi-recursive network is modified by the inclusion of multiple copies of the top-ranked feature, $d_{-}d_{+}^{a}$. These patterns are so strong that an algorithm to generate good $p <\approx .3$ networks recursively would be computationally far cheaper than the hill-climbing method.

The running time of the search over part-sharing trees on a conventional serial computer (a Pyramid

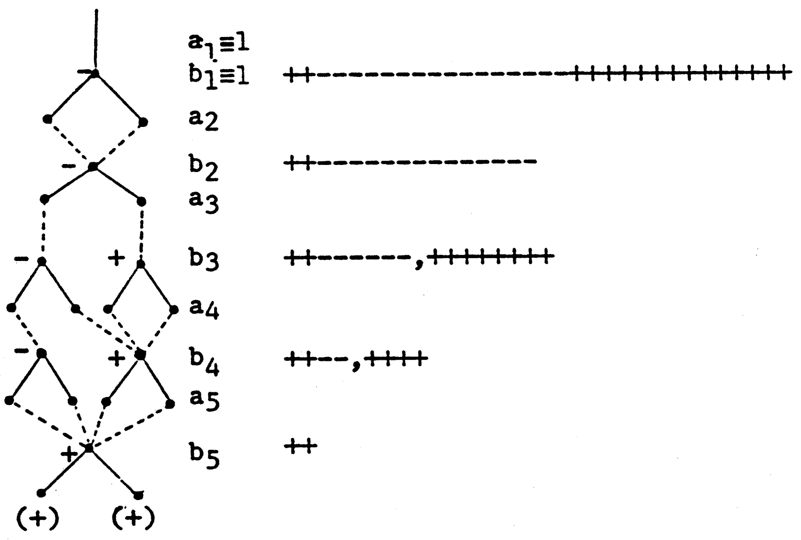

Figure 1. A part-sharing tree from a computed feature matrix for $N = 32, A = 10, p = .5$. Solid lines show concatenation of feature vectors and dotted lines indicate multiple use of one feature vector. Signs attached to the tree nodes indicate concatenation with $(-)$ and without $(+)$ feature negation. The bottom-most signs are always taken to be positive, to reduce the redundancy in the number of part-sharing trees which represent one feature. The middle column records the number of nodes at each level in the structure; case illustrated has $a_2 = 2$ and $b_2 = 1$. The feature vectors corresponding to the nodes at a given level in the tree are shown on the right.

90X) is much faster than that of the Walsh function beam search described earlier, and a factor of 10 slower than hill-climbing (measured times were 120 sec, 43.2 sec, and 3.5 sec respectively for $(N, A, p) = (16,8,.1)$; 356 sec, 939 sec, and 24.2 sec respectively for $(N, A, p) = (32,10,.03)$; and 900 sec, unmeasured and 103 sec respectively for $(N, A, p) = (64,12,.3)$). We have several comments on this timing data. The Walsh function beam search is impractically slow, but the resulting networks are so highly patterned that one can generalize them to all larger sized networks and thereby produce large networks far more quickly than a hill-climbing procedure could. However, our real interest is see how this generalization could automatically come out of a more fully recursive description of the network, without human intervention. The slow search

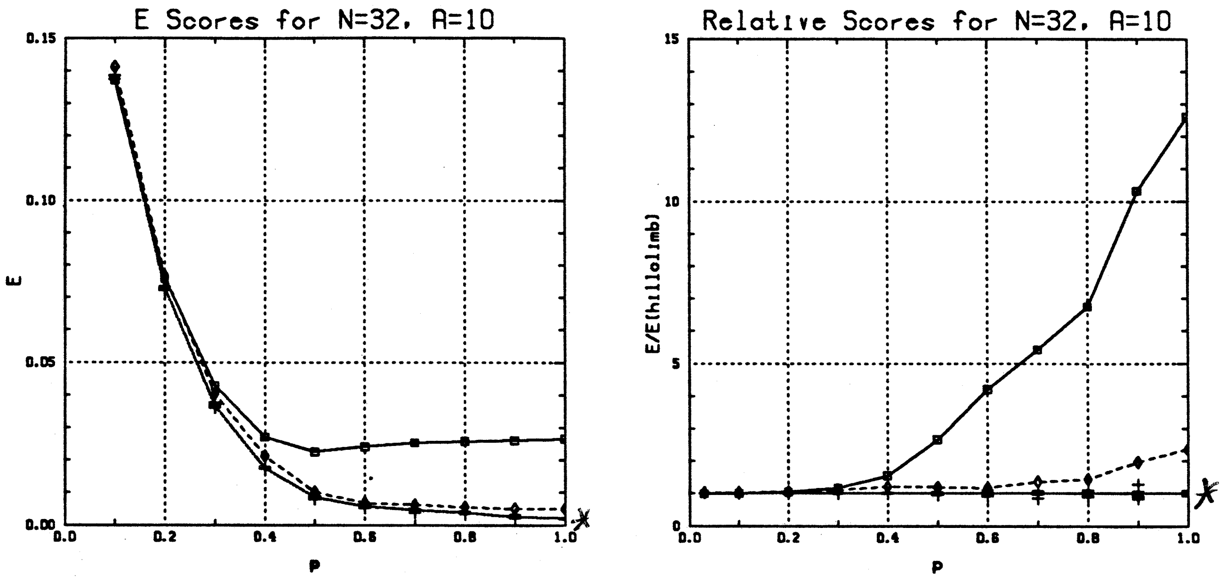

Figure 2. Relative and absolute E scores (see equation (1)) for the continuous encoding problem with $N = 32$ and $A = 2\log_2 N$. The three optimization methods compared are hillclimbing (dotted lines), beam search over features constructed by duplication with and without negation (solid lines), and beam search over part-sharing trees (dashed lines). These methods are discussed further in the text. (a) Absolute E scores for $N = 32$. (b) Scores relative to the average of five hillclimbing runs, for $N = 32$. The corresponding plots for $n = 16$ are almost indistinguishable from these ones.

over part-sharing trees is 10 times slower than hill-climbing, but is far faster than the slow search over Walsh function networks and produces better networks. The question of the orderliness or pattern of the optimal part-sharing networks remains to be understood and is the key to any automatic generalization procedure.

Aside from the important question of the presence or absence of generalizable patterns in the part-sharing tree feature descriptions, the most pressing problem for investigation is probably the replacement of the beam search by which features $F_{\alpha*}$ are brought together to make a feature matrix F with a fully recursive description of F. We speculate that the two problems may have to be solved jointly.

Conclusion

By recursively building up a neural network to perform the continuous encoding task, one can obtain networks described by so few parameters (so highly structured) that a unique generalization to all larger networks and problem sizes is suggested. When the parameter p in the continuous encoding task is less than

feature vector	description	rank	blocks
++++++++++++++++++++++++++++++++	$d_+d_+d_+d_+d_+$	∞	1
++++++++++++++++----------------	$d_-d_+d_+d_+d_+$	1	2
++++++++----------------++++++++	$d_-d_-d_+d_+d_+$	2	3
++++++++--------++++++++--------	$d_+d_-d_+d_+d_+$	3	4
++++--------++++++++--------++++	$d_+d_-d_-d_+d_+$	4	5
++++--------++++----++++++++----	$d_-d_-d_-d_+d_+$	5	6
++++----++++--------++++----++++	$d_-d_+d_-d_+d_+$	6	7
++++----++++----++++----++++----	$d_+d_+d_-d_+d_+$	7	8
++----++++----++++----++++----++	$d_+d_+d_-d_-d_+$	8	9
++----++++----++--++++----++++--	$d_-d_+d_-d_-d_+$	9	10
++----++--++++----++++--++----++	$d_-d_-d_-d_-d_+$	10	11

Figure 3. Highly patterned optimal features for $p = .01$. Except for the top feature, which is chosen last, the top A features are chosen to form a feature matrix. The pattern of $d_\pm$ operations is best seen by reading down vertical columns of aligned operators; the signs alternate in pairs and successive powers of two. (The feature $d_+d_+d_+d_+d_+$ is placed first to emphasize this pattern, though it is the exception.) The ranking also corresponds to the number of same-sign blocks in the feature; coarse features are chosen first.

$\approx .3$, the resulting network performs well. For most of the range in p, we can obtain good performance by considering less rigid semi-recursive descriptions whose proper generalization to large networks is not yet clear. These networks involve the use of shared standard "parts" and the re-use of previously tested "templates" for large-scale network organization; the templates, at least, are learned from experience with smaller versions of the same encoding task.

References

1. G. Hinton and T. Sejnowski, Proc. 5th Annual Conference of the Cognitive Science Society, 1983.
2. A. Lapedes and R. Farber, Los Alamos preprint LA-UR-85-4037, 1985.
3. K. Wilson and J. Kogut, Phys. Rept. 12C Number 2, 1974.